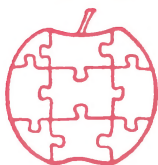


Apple

\$1.50



Assembly

Line

Volume 3 -- Issue 12

September, 1983

In This Issue...

Jump Vectoring	2
Using QUICKTRACE with the S-C Assembler.	8
Generate Machine Code with Applesoft	10
Amper-Monitor.	14
Yet Another New Version of DOS 3.3	16
Base Address Calculation	18
Saving Source Files for Apple's Mini-Assembler	21
Generic Screen Dump.	22
New CATALOG Interrupt.	26
80 Column ASCII Monitor Dump	27

65C02 Notes

We now have a sample from Rockwell, and it shares the problem of not working in an older Apple. It's running just fine in the //e, but it doesn't work in the][+. Rockwell's distributor says that regular delivery is now scheduled for November. Sigh....

There's a bug in the 65C02 chips! Among the new features are several new addressing modes for the BIT instruction, including BIT #immediate.

The BIT instruction actually does two operations:

- 1) It ANDs together the Accumulator and the specified memory byte, and sets the Zero flag according to the result.
- 2) It sets the Overflow and Negative flags to the values of bits 6 and 7 of the memory byte.

Well, the BIT #immediate instruction does not do step two; it only modifies the Zero flag. The other new address modes for BIT behave correctly. BIT #\$40 sure would have come in handy for a SEV (SEt oVerflow flag) instruction.

As always, we'll keep you posted.

Jump Vectoring.....Bob Sander-Cederlof

Applesoft has a statement which allows branching according to a computed index:

```
ON X GO TO 100,200,300,400
```

Integer BASIC has a different method, simply allowing the line number after a GOTO, THEN, or GOSUB to be a computed value:

```
GO TO X*100
```

Most other languages have some technique for vectoring to one of a series of places based on the value of a variable. Modern languages like Pascal have a CASE statement, which can combine a comparison step.

```
case PIECE of
  Pawn : ...;
  Knight : ...;
  Bishop : ...;
  Rook : ...;
  Queen : ...;
  King : ...;
end
```

I frequently find myself building various schemes to handle the CASE statement in assembly language. For example, I might accept a character from the keyboard and then compare it to a series of legal inputs, and branch accordingly to process the input.

One common way involves a series of CMP BEQ pairs, like this:

```
JSR GETCHAR
CMP #$81      control-A?
BEQ ...      yes
CMP #$84      control-D?
BEQ ...      yes
CMP #$8D      return?
BEQ ...      yes
et cetera
```

If there are not too many cases, and if the processing routines are not too far away for the BEQs to reach, this is a good way to do the job. If the routines are bigger, and therefore tend to be too far away (causing RANGE ERRORS at assembly time), I might string together CMP BNE pairs instead:

```
JSR GETCHAR
CMP #$81      control-A?
BNE TRY.D     no, try ctrl-D
```

<code to process ctrl-A here>

```
TRY.D  CMP #$84      control-D?
      BNE TRY.M     no, try return
```

S-C Macro Assembler (the best there is!).....\$80.00
 S-C Macro Assembler Version 1.1 Update.....\$12.50

S-C Cross Reference Utility.....\$20.00
 S-C Cross Reference Utility with Complete Source Code.....\$50.00

S-C Word Processor (the one we use!).....\$50.00
 With fully commented source code. Needs S-C Macro Assembler.

Applesoft Source Code on Disk.....\$50.00
 Very heavily commented. Requires Applesoft and S-C Assembler.

ES-CAPE: Extended S-C Applesoft Program Editor.....\$60.00

AAL Quarterly Disks.....each \$15.00
 Each disk contains all the source code from three issues of "Apple
 Assembly Line", to save you lots of typing and testing time.
 QD#1: Oct-Dec 1980 QD#2: Jan-Mar 1981 QD#3: Apr-Jun 1981
 QD#4: Jul-Sep 1981 QD#5: Oct-Dec 1981 QD#6: Jan-Mar 1982
 QD#7: Apr-Jun 1982 QD#8: Jul-Sep 1982 QD#9: Oct-Dec 1982
 QD#10: Jan-Mar 1983 QD#11: Apr-Jun 1983 QD#12: Jul-Sep 1983

Double Precision Floating Point for Applesoft.....\$50.00
 Provides 21-digit precision for Applesoft programs.
 Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research).....\$79.00
 Full Screen Editor for S-C Macro Assembler (Laumer Research).....\$49.00

The Visible Computer: 6502 (Software Masters).....(reg. \$50.00) \$45.00
 Amper-Magic (Anthro-Digital).....(reg. \$75.00) \$67.50
 Amper-Magic Volume 2 (Anthro-Digital).....(reg. \$35.00) \$30.00
 Quick-Trace (Anthro-Digital).....(reg. \$50.00) \$45.00
 DISASM Dis-Assembler (RAK-Ware).....\$30.00

Blank Diskettes.....package of 20 for \$45.00
 (Premium quality, single-sided, double density, with hub rings)
 Small 3-ring binder with 10 vinyl disk pages and disks.....\$36.00
 Vinyl disk pages, 6"x8.5", hold one disk each.....10 for \$6.00
 Diskette Mailing Protectors.....10-99: 40 cents each
 100 or more: 25 cents each

ZIF Game Socket Extender.....\$20.00
 Shift-Key Modifier.....\$15.00
 Lower-Case Display Encoder ROM.....\$25.00

Only Revision level 7 or later Apples.
 STB-80 80-column Display Board (STB Systems).....(\$249.00) \$225.00
 STB-128 128K RAM Card (STB Systems).....(\$399.00) \$350.00

Grappler+ Printer Interface (Orange Micro).....(\$175.00) \$150.00
 Bufferboard 16K Buffer for Grappler (Orange Micro).....(\$175.00) \$150.00
 Buffered Grappler+ NEW!! Interface and 16K Buffer.....(\$239.00) \$200.00

Books, Books, Books.....compare our discount prices!
 "The Apple][Circuit Description", Gayler.....(\$22.95) \$21.00
 "Enhancing Your Apple II, vol. 1", Lancaster.....(\$17.95) \$17.00
 "Incredible Secret Money Machine", Lancaster.....(\$7.95) \$7.50
 "Micro Cookbook, vol. 1", Lancaster.....(\$15.95) \$15.00
 "Micro Cookbook, vol. 2", Lancaster.....(\$15.95) \$15.00
 "Beneath Apple DOS", Worth & Lechner.....(\$19.95) \$18.00
 "Bag of Tricks", Worth & Lechner, with diskette.....(\$39.95) \$36.00
 "Apple Graphics & Arcade Game Design", Stanton.....(\$19.95) \$18.00
 "Assembly Lines: The Book", Roger Wagner.....(\$19.95) \$18.00
 "What's Where in the Apple", Second Edition.....(\$24.95) \$23.00
 "What's Where Guide" (updates first edition).....(\$9.95) \$9.00
 "6502 Assembly Language Programming", Leventhal.....(\$18.95) \$18.00
 "6502 Subroutines", Leventhal.....(\$17.95) \$17.00
 Add \$1.50 per book for US postage. Foreign orders add postage needed.

Whatever Else You Need.....Call for Our Low Prices

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
 *** (214) 324-2050 ***
 *** We accept Master Card, VISA and American Express ***

```

<code for ctrl-D here>
TRY.M  CMP #$8D          return?
      BNE ... et cetera

```

```

<code for ctrl-M here>

```

The trouble with the latter way is that programs get strung all over the place, and become very difficult to follow. Unstructured, some would say. The structure is really there, because we are just implementing a CASE statement; however, assembly language code over a sheet of paper long LOOKS unstructured, no matter what it is implementing. And once a programmer gets his CASE statement spread over several sheets of paper, the temptation to begin making a "rat's nest" out of it can be overwhelming.

I prefer to put things into nice neat data tables. Back in the August 1982 issue of AAL I presented a "Search and Perform" subroutine to handle a table like this:

```

.DA #$81,CTRL.A-1
.DA #$84,CTRL.D-1
.DA #$8D,RETURN-1
etc.

```

The table consists of three bytes per line, the first byte being the CASE value, and the other two being the address of the processing routine.

Another method is handy when the variable has a nice numeric range. For example, what if I have processing routines for every possible control character from ctrl-A through ESC? That is ASCII codes \$81 through \$9B. If I subtract \$81, I get a value from 0 through 26 (decimal). If I then multiply the value by three, and add it to a base address, and store the result into another variable, and JMP indirect, I can access a series of JMPs to each processing routine:

```

CASE      JSR GETCHAR
          SEC
          SBC #$81
          CMP #27
          BCS ...ERROR, NOT IN RANGE
          STA ADDR          TIMES THREE
          ASL
          ADC ADDR
          ADC #TABLE        PLUS TABLE BASE ADDRESS
          STA ADDR
          LDA #0
          ADC /TABLE
          STA ADDR+1
          JMP (ADDR)
ADDR      .BS 2
TABLE     JMP CTRL.A
          JMP CTRL.B
          .
          .
          JMP ESCAPE

```

Note that if we got to the CASE program by doing a JSR CASE, then each processing routine can do an RTS to return to the main line program. This makes our CASE look like it is doing a series of JSR's instead of JMP's.

We can shave bytes off the above technique by only keeping the address in TABLE, without all the JMP opcodes. Then the variable only needs to be multiplied by two instead of three. We will have to use the doubled variable for an index to pick up the address in the table and put it into ADDR:

```

CASE      JSR GETCHAR
          SEC
          SBC #$81
          CMP #27
          BCS ...ERROR, NOT IN RANGE
          ASL      DOUBLE THE INDEX
          TAX
          LDA TABLE,X
          STA ADDR
          LDA TABLE+1,X
          STA ADDR+1
          JMP (ADDR)
ADDR      .BS 2
TABLE     .DA CTRL.A
          .DA CTRL.B
          .
          .DA ESCAPE

```

I don't recommend self-modifying code, but I still use it sometimes. If you want to save two more bytes above, then you can store the jump address directly into the second and third bytes on a direct JMP instruction:

```

          LDA TABLE,X
          STA ADDR+1
          LDA TABLE+1,X
          STA ADDR+2
ADDR      JMP 0

```

A much better way involves pushing the processing routine address onto the stack, and using an RTS to branch to the pushed address. Since RTS adds 1 to the address on the stack before branching, we have to push the address-1:

```

          LDA TABLE+1,X
          PHA      HIGH BYTE FIRST
          LDA TABLE,X
          PHA
          RTS
TABLE     .DA CTRL.A-1
          .DA CTRL.B-1
          .
          .DA ESCAPE-1

```

Note that this method not only is not self-modifying, it also is a few bytes shorter and a tad faster.

All this is only necessary because the designers of the 6502 did not give us a JMP (addr,X) instruction. If they had, we could do it like this:

```

        JSR GETCHAR
CASE     SEC
        SBC #$81
        CMP #27
        BCS ...ERROR
        ASL                DOUBLE FOR INDEX
        TAX
        JMP (TABLE,X)
TABLE    .DA CTRL.A, CTRL.B,...,ESCAPE
```

Then the hardware would add the doubled character offset (0,2,4,...52 for ctrl-A thru ESC) to the base address of the table, pick up the address from the table, and jump to the corresponding processing routine.

Since that would be so nice, and the designers agreed, the new 65C02 chip has it! So if you know you are writing for a 65C02, and don't EVER intend to run in a plain 6502, you can use the JMP (TABLE,X).

It would also be nice to have JSR (TABLE,X), but you can simulate that by calling CASE with a JSR. Or in other situations, you might merely do it this way:

```

        JSR CALL
        .
        .
CALL     JMP (TABLE,X)
```

Sometimes it so happens that your program can be arranged so that all the processing routines are in the same memory page. Then there is no need to store the high byte of the address in the table, right? Steve Wozniak thought this way, and you can see the result in the Apple monitor at \$FFBE and following:

```

TOSUB   LDA #$FE                HIGH BYTE OF ALL ADDRESSES
        PHA
        LDA SUBTBL,Y
        PHA
ZMODE    LDY #0
        STY MODE
        RTS
        .
        .
SUBTBL   .DA #BASCONT-1          CTRL-C
        .DA #USR-1              CTRL-Y
        .DA #BEGZ-1             CTRL-E
        .
        .
        .DA #BLANK-1            BLANK
```



PERSONAL ROBOTS

Peripherals and Software for Personal Robots

MICROMATION introduces a new line of products designed to enhance Heathkit's® HERO-I robot. These products eliminate the current problem of having to hand code programs into the robot. They provide the software and hardware necessary to allow rapid development of long, meaningful programs for HERO on your personal computer. Our software will also provide you with examples of advanced programming techniques and show you how to generate artificial intelligence programs in machine language.

HERO MEMCOM BOARD

This product provides a means to develop programs for the robot using a personal computer, and expands the robot's memory with an additional 30K of RAM. A large wire-wrapping area is also available on the board for user experimentation. The product includes:

- Two 8-bit bi-directional parallel ports with handshaking lines for superfast data transfers between the robot and a computer (connects directly to our APPLE-HERO COMMUNICATOR board), plus two 16-bit timers.
- An RS232 serial port for two-way communications between the robot and any computer having an RS232 serial port.
- Serial communications software in an onboard EPROM which allows uploading and downloading of programs via the RS232 port.
- Sixteen 2K RAM chips in sockets so 2K EPROM chips may be substituted, if desired. "Memory Protect" and "Ram Supply" are used for each RAM chip so programs will not be lost when robot enters "sleep" mode.
- A large (2" x 5") wire-wrap area with CPU address, data and control lines, $\pm 5V$, ground, and others readily available.
- All hardware to mount board on rear door of robot and to connect board to the robot's CPU board.
- Complete instruction manual and schematics.

PRICE \$295.00

APPLE-HERO COMMUNICATOR

This product provides the hardware and software necessary to implement two-way high speed parallel communications between an APPLE® computer and a HERO-I robot equipped with our HERO MEMCOM BOARD. The product includes:

- A peripheral card for an APPLE that contains two 8-bit parallel ports with handshaking lines, and two 16-bit timers.
- A ribbon cable (about 5 feet long) with connectors for connecting the parallel ports on the card to those on the HERO MEMCOM BOARD.
- Data transfer software for the APPLE board and for the HERO MEMCOM BOARD burned into two 2716 EPROMs. These programs provide ultra fast two-way communications for rapid downloading and uploading of programs.
- A disk containing heavily commented 6808 and 6502 source codes for the communications software. These source codes are compatible with the S-C MACRO ASSEMBLER and the S-C 6800 CROSS ASSEMBLER available for the APPLE from the S-C SOFTWARE CORPORATION. Programs developed using the S-C 6800 CROSS ASSEMBLER can be immediately downloaded and run on the robot using this product and our HERO MEMCOM BOARD.

PRICE \$159.00

STORYTELLER

This is an artificial intelligence program that turns HERO into a teller of galactic tales. The program uses an advanced self-programming technique which allows the robot to speak randomly generated, but grammatically correct, sentences concerning its space voyages. The program is both amazing and amusing. After listening to the robot for a while one gets the impression that it has just returned from a space battle with alien invaders.

PRICE: Tape (machine code) \$15.00

Disk (source code) \$25.00

ROBOT LANGUAGE MACRO

This program consists of an S-C 6800 CROSS ASSEMBLER macro source code which completely implements the HERO Robot Language opcodes. This macro provides the user with a very fast means of developing programs for the robot when used in conjunction with our APPLE-HERO COMMUNICATOR and HERO MEMCOM BOARD. The S-C 6800 CROSS ASSEMBLER is required in order to use this product.

PRICE \$35.00

6808 CROSS DISASSEMBLER

This program allows 6808 machine code resident in an APPLE's memory to be disassembled in a format similar to that used by the APPLE monitor. Disassembly can be performed optionally into text files for later EXECing into the 6800 CROSS ASSEMBLER. Machine code resident in HERO's memory (like the robot's monitor) can be made into source codes by first transferring the code to the APPLE's memory (using the APPLE-HERO COMMUNICATOR and HERO MEMCOM BOARD) and then disassembling to a text file. This text file can then be EXECed into the S-C 6800 CROSS ASSEMBLER and meaningful comments added. Source code for the program is also available.

PRICE: Program only \$35.00

Program and source code \$55.00

Send check or money order to:

MICROMATION INC.

9104 Red Branch Rd.
Columbia, MD 21045



For information call:

(301) 730-1237

9 am-5pm Monday through Friday
MasterCard & Visa welcome

Steve also used this technique inside the SWEET-16 interpreter. You can see the code at \$F69E through \$F6C6 in the Integer BASIC ROM or RAM image.

If the routines are not necessarily all in one page, but are all within one 256-byte range, you can add an offset from the table to a known starting address.

Here is a method I would NEVER use, but it is cute, and short:

```
        LDA TABLE,X      X IS CALCULATED INDEX
        STA BRANCH+1      INTO BCC INSTRUCTION
        CLC               make branch always...
BRANCH  BCC BRANCH        2ND BYTE GETS FILLED IN
BASE    .EQ *
        ...
        ...all the routines here
        ...
TABLE   .DA #CTRL.A-BASE
        .DA #CTRL.B-BASE
        etc.
```

The table has pre-computed relative offsets from BASE, so that the values can be plugged directly into the BCC instruction. This is a fast and short technique, but somehow it scares me to think about self-modifying code. If you need it, go ahead and use it!

Using QUICKTRACE with S-C Assembler.....Bob Urschel
Valparaiso, IN

I wanted to use QUICKTRACE in conjunction with the S-C Assembler without having QUICKTRACE interfere with either my source file or any object code generated. Since I always use the LC version of the assembler, I modified the HELLO program on the S-C assembler disk as follows:

```
10 HOME:PRINT "LOADING QUICKTRACE..."
20 POKE 40192,211:POKE40193,142:CALL42964
30 PRINT CHR$(4)"BLOAD QUICKTRACE,A$8F00"
40 PRINT:PRINT "LOADING S-C ASSEMBLER..."
50 VTAB24:POKE34,23:PRINTCHR$(4)"EXEC LOAD LCASM"
60 END
```

Line 20 in the HELLO program modifies the location of the DOS buffers by \$E00 bytes to make room for the QUICKTRACE program. After running the HELLO program, when the S-C prompt appears and BEFORE loading any S-C source files, enter:

```
: $8F00G <return>
```

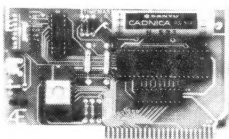
This initializes QUICKTRACE.

I also changed the address at MON\$ (from within QUICKTRACE) to MON\$=D003 so when I press M from single-step mode, I return to the S-C Assembler with my source file intact.

APPLIED ENGINEERING

THE BEST PERIPHERALS FOR THE BEST COMPUTER

The TIMEMASTER Finally a clock that does it ALL!



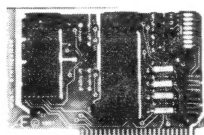
- Designed in 1983 using I.C. technologies that simply did not exist when most other Apple clocks were designed.
- Just plug it in and your programs can read the year, month, date, day, and time to 1 millisecond! The only clock with both year and ms.
- Powerful 2K ROM driver — No clock could be easier to use.
- Full emulation of most other clocks, including Mountain Hardware's Appleclock (but you'll like the TIMEMASTER mode better).
- Basic, Machine Code, CP/M and Pascal software on 2 disks!
- Eight software controlled interrupts so you can execute two programs at the same time. (Many examples are included)
- On board timer lets you time any interval up to 48 days long down to the nearest millisecond.

The TIMEMASTER includes 2 disks with some really fantastic time oriented programs (over 25) plus a DOS dater so it will automatically add the date when disk files are created or modified. This disk is over a \$200.00 value alone — we give the software others sell. All software packages for business, data base management and communications are made to read the TIMEMASTER.

If you want the most powerful and the easiest to use clock for your Apple, you want a TIMEMASTER.

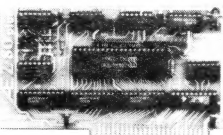
PRICE \$129.00

Super Music Synthesizer



- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Will play songs written for ALF synthesizer (ALF software will not take advantage of all the features of this board. Their software sounds the same in our synthesizer.)
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE \$159.00



Z-80 PLUS!

- **TOTALLY** compatible with **ALL** CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- All new 1983 design incorporates the latest in I.C. technologies.

COMING SOON: The Z-80 Plus for the Apple III

PRICE \$139.00

Viewmaster 80

There used to be about a dozen 80 column cards for the Apple, now there's only **ONE**.

- **TOTALLY** Videx Compatible
- 80 characters by 24 lines, with a sharp 7x9 dot matrix
- On-board 40/80 soft video switch with manual 40 column override
- Fully compatible with **ALL** Apple languages and software — there are **NO** exceptions
- Low power consumption through the use of CMOS devices
- All connections on the card are made with standard video connectors, no cables are soldered to the board
- All new 1983 design (using a new Microprocessor based C.R.T. controller)

JUST COMPARE!

	VIEWSER 169	SUP'RTERM 175	WIZARD80 245	VISION80 375	OMNIVISION 295	VIEWMAX80 219	SMARTERM 360	VIDEOTERM 345	VIEWMASTER 80	OTHERS
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES
80 COLUMN	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, EasyWriter, Apple Writer II, Visicalc, and many others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

PRICE \$169.00

MemoryMaster IIe

- Expands your Apple IIe to 192K memory
- Provides an 80 column text display
- Compatible with all Apple IIe 80 column and extended 80 column card software. (Same physical size as Apple's 64K card)
- Available in 64K and 128K configurations
- Bank select LED's for each 64K bank
- Permits your IIe to use the new double high resolution graphics
- Automatically expands Visicalc to 95K storage in 80 columns! The 64K configuration is all that's needed, 128K can take you even higher.

128K RAM Card

- Complete documentation included, we show you how to use all 128K. If you already have Apple's 64K card, just order the MEMORYMASTER with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

MemoryMaster with 128K

\$249

Upgradeable MemoryMaster with 64K

\$169

Non-Upgradeable MemoryMaster with 64K

\$149

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in APPLE IIe, IIx and Franklin (except MemoryMaster). Applied Engineering also manufactures a full line of data acquisition and control products for the Apple: A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY**.

All Orders Shipped Same Day, Texas Residents Add 5% Sales Tax. Add \$10.00 If Outside U.S.A. Dealer Inquiries Welcome.

Send Check or Money Order to:
APPLIED ENGINEERING
P.O. Box 470301
Dallas, TX 75247

Call (214) 492-2027
7am. to 11p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome

Generate Machine Code with Applesoft.....Bob Sander-Cederlof

Apparently nobody picked up my challenge at the end of the article about Charlie Putney's faster spiral screen clear program (August 1983 AAL, page 16). I suggested someone write a program in Applesoft which would in turn construct a machine language screen clear.

Nobody else did it, so I did. And whether you are interested in fancy ways to clear the screen or not, the techniques I used may be put to other uses.

The task of building a screen clear program can be divided into two parts. First, generate the memory addresses of the 960 cells on the screen, in the order (or path) that the spiral shift will follow. Second, using that table of addresses, generate the 959 pairs of LDA and STA instructions necessary to move the screen one position along the spiral path. There is really a third part: to generate the necessary prologue and postlogue instructions to make those 959 LDA-STA pairs be executed 960 times, and to clear the vacated byte at the tail end of the spiral path.

After trying various ways to understand the spiral path, I arrived at a table-driven approach. I put the table into data statements (lines 3000-3110 below), and made a simple loop to generate the 960 addresses (lines 100-150).

You might notice that the twelve lines of data correspond very closely to the parameters on Charlie Putney's macro calls. After I typed in the twelve lines, I noticed a definite pattern. I could have used only the first line of data, and computed the others by a simple algorithm: increment each value smaller than 13, and decrement each value 13 or larger. Well, no program is ever finished....

Once the 960 addresses are stored in array A\$(0) through A\$(959), I proceed to generate machine language code. Line 180 does it all, with the help of four simple subroutines. Then line 190 rings the bell, and line 200 calls the machine language program just generated for a fast two-and-a-half second demonstration.

During the address array building process, I fill up the screen with the letters U, D, L, and R. These show the direction (up, down, left, and right) which a given character will be shifted along the spiral path. The directions are just the opposite from the order in which the letters are displayed, because I generate the address list backwards (from head to tail).

During the generation of the machine language program, which takes about two minutes, I toggle the tail end character between normal to inverse video. This gives you something to watch for those llooonnggg two minutes.

The generation process is broken into four parts, represented by four subroutines at 5000, 5100, 5200, and 5300.

GOSUB 5000 generates a four byte prologue, starting at memory address \$2710, or 10000. The code looks like this:

```
LDX /-960
LDY #-960
```

Actually, not -960, but -960/S. S gives a step size. Sidestepping a little from the main discussion, let me tell you about S.

Don Lancaster called last week to talk about a few things with Bill, and passed on the results of his experiments with Charlie's program. He noted that the video refresh rate is 60 times per second, and that a 7.5 second screen clear moves a little more than two steps for each frame time. Therefore you don't really SEE each step. Therefore the screen clear routine could move each character two steps ahead at a time with the same smooth effect on the screen, but clearing the screen in half the time. Or three steps, clearing in one third the time. The variable S in my program lets you experiment with the number of steps each character moves during each pass. As listed, S=3, so the screen clears in 2.5 seconds.

GOSUB 5100 generates the requisite number of LDA-STA pairs to move the screen one step of size S along the spiral path.

GOSUB 5200 generates the instructions to clear the bytes at the tail end of the spiral. If S=3, you will get:

```
LDA #$A0          BLANK
STA $636
STA $635
STA $634
```

GOSUB 5300 generates the end-of-loop code:

```
INY
BNE LP
INX
BNE LP
RTS
LP    JMP 10004
```

The screen need not necessarily be cleared to all blanks. By changing the value POKed in the second part of line 5210 you can fill with all stars, or all white, or whatever.

Another interesting option occurs to me. Given a table in the A% array of all the screen addresses, in any arrangement that suits my fancy, I can clear the screen in 2.5 to 7.5 seconds by shifting the screen along that particular path. It could be random, spiral, kaleidoscopic, or whatever.

There are so many other things I could explain about this little program, I hardly know where to stop. I think I'll stop here, and leave the rest for your own rewarding investigation and analysis.

```

100 TEXT : HOME : DIM A$(1000)
105 N = 0
110 READ X,YB,YT: GOSUB 1000
120 READ Y,XL,XR: GOSUB 1200
130 READ X,YT,YB: GOSUB 1100
140 READ Y,XR,XL: GOSUB 1300
150 IF N < 960 THEN 110
160 REM
      BUILD MACHINE LANGUAGE SPIRAL SHIFT

170 S = 3
180 GOSUB 5000: GOSUB 5100: GOSUB 5200: GOSUB 5300
190 CALL - 1054: REM RING BELL
200 CALL 10000: END
500 REM
      POKE ADDRESS

510 AH = INT (A / 256):AL = A - AH * 256: POKE L + 1,AL: POKE L + 2,AH:L
  = L + 3: POKE 1588,256 - PEEK (1588): RETURN
1000 REM
      MOVE DOWN COLUMN X FROM YB TO YT

1010 C$ = "D": FOR Y = YB TO YT STEP - 1
1020 VTAB Y + 1: GOSUB 2000: NEXT : RETURN
1100 REM
      MOVE UP COLUMN X FROM YT TO YB

1110 C$ = "U": FOR Y = YT TO YB: VTAB Y + 1: GOSUB 2000: NEXT : RETURN
1200 REM
      MOVE LEFT ROW Y FROM XL TO XR

1210 C$ = "L": VTAB Y + 1: FOR X = XL TO XR: GOSUB 2000: NEXT : RETURN
1300 REM
      MOVE RIGHT ROW Y FROM XR TO XL

1310 C$ = "R": VTAB Y + 1: FOR X = XR TO XL STEP - 1: GOSUB 2000: NEXT :
  RETURN
2000 REM
      POST ADDRESS

2010 A = PEEK (40) + PEEK (41) * 256 + X:A$(N) = A:N = N + 1: POKE A, ASC
  (C$) + 128
2020 RETURN
3000 DATA 0,23,0, 0,1,39, 39,1,23, 23,38,1
3010 DATA 1,22,1, 1,2,38, 38,2,22, 22,37,2
3020 DATA 2,21,2, 2,3,37, 37,3,21, 21,36,3
3030 DATA 3,20,3, 3,4,36, 36,4,20, 20,35,4
3040 DATA 4,19,4, 4,5,35, 35,5,19, 19,34,5
3050 DATA 5,18,5, 5,6,34, 34,6,18, 18,33,6
3060 DATA 6,17,6, 6,7,33, 33,7,17, 17,32,7
3070 DATA 7,16,7, 7,8,32, 32,8,16, 16,31,8
3080 DATA 8,15,8, 8,9,31, 31,9,15, 15,30,9
3090 DATA 9,14,9, 9,10,30, 30,10,14, 14,29,10
3100 DATA 10,13,10, 10,11,29, 29,11,13, 13,28,11
3110 DATA 11,12,11, 11,12,28, 28,12,12, 12,27,12
5000 REM
      COMPILE PROLOGUE

5010 T = 65536 - 960 / S:TH = INT (T / 256):TL = T - TH * 256
5020 POKE 10000,162: POKE 10001,TH
5030 POKE 10002,160: POKE 10003,TL
5040 RETURN
5100 REM
      COMPILE LDA-STA PAIRS

5110 L = 10004: FOR I = 0 TO 957: POKE L,173:A = A$(I + S): GOSUB 500
5120 POKE L,141:A = A$(I): GOSUB 500: NEXT
5130 RETURN
5200 REM
      COMPILE CLEAR S BYTES

5210 POKE L,169: POKE L + 1,160:L = L + 2
5220 FOR I = 1 TO S: POKE L,141:A = A$(960 - I): GOSUB 500: NEXT
5230 RETURN
5300 REM
      COMPILE POSTLOGUE

5310 FOR I = 0 TO 9: READ A: POKE L + I,A: NEXT
5320 RETURN
5350 DATA 200,208,4,232,208,1,96,76,20,39

```

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these **FEATURES**:

Single-Step mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.

Trace mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.

Background mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.

QUICKTRACE allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.

Two optional display formats can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.

QUICKTRACE is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.

QUICKTRACE is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.

QUICKTRACE is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.

QUICKTRACE is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program

QUICKTRACE

\$50

Is a trademark of Anthro-Digital, Inc.

Copyright © 1981

Written by John Rogers

See these programs at participating Computerland and other
fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

Amper-Monitor.....Bob Sander-Cederlof

It would be nice to be able to use monitor commands from within Applesoft, both in direct commands and within running Applesoft programs. At least Kraig Arnett, from Homestead, Florida, thinks so.

I agree, and so I whipped out another handy-dandy &-subroutine for just that purpose. I call it Amper-Monitor. You can install it by BRUNning it from a binary file, or by adding some POKes to your Applesoft program. My listing shows it residing at the ever popular \$300 address, but it can be reassembled to run anywhere. Just remember to connect it properly to the Ampersand Vector.

Once Amper-Monitor is installed and hooked to the ampersand vector, you call it by typing an ampersand, a quotation mark, and a monitor command. Here is a sample program showing some uses of the Amper-Monitor.

```
100 FOR I = 768 TO 855
110 READ D : POKE I,D : NEXT
120 CALL 768

130 &"300.357
140 &"380:12 34 56 78 9A BC DE F0
150 &"FBE2G
160 &"300L 380.387

200 DATA 169,11,141,246,3,169,3,141,247,3,96
210 DATA 201,34,208,70,32,177,0,160,0,177,184,201,0
220 DATA 240,8,9,128,153,0,2,200,208,242,169,141
230 DATA 153,0,2,152,24,101,184,133,184,144,2,230
240 DATA 185,32,199,255,32,167,255,132,52,160,23
250 DATA 136,48,23,217,204,255,208,248,192,21,240
260 DATA 8,32,190,255,164,52,76,52,3,32,197,255
270 DATA 76,0,254,76,201,222
```

Why did I choose to require the quotation mark after the ampersand? Because normally Applesoft would parse the line, eliminating blanks, changing DEF to a token instead of three hex digits, using ":" to end a line, and so on. Using the "-mark prevents all this, leaving the line in raw ASCII form. Here is a listing of the program in assembly language:

```
1000 *SAVE S.AMPER.MONITOR
1010 *-----
1020 *      &-MONITOR COMMANDS
1030 *-----
0031- 1040 MON.MODE .EQ $31
0034- 1050 MON.YSAV .EQ $34
00B8- 1060 TXTPTR .EQ $B8 AND B9
0200- 1070 MON.BUFFER .EQ $200
03F5- 1080 AMPERSAND.VECTOR .EQ $3F5
1090 *-----
00B1- 1100 AS.CHRGET .EQ $00B1
DEC9- 1110 AS.SYNERR .EQ $DEC9
FE00- 1120 MON.BL1 .EQ $FE00
FFA7- 1130 MON.GETNUM .EQ $FFA7
FFBE- 1140 MON.TOSUB .EQ $FFBE
FFC7- 1150 MON.ZMODE .EQ $FFC7
FFCC- 1160 MON.CHRTBL .EQ $FFCC
```

```

1170 *-----
1180 .OR $300
1190 *-----
0300- A9 0B 1200 SETUP LDA #AMPER.MONITOR
0302- 8D F6 03 1210 STA AMPERSAND.VECTOR+1
0305- A9 03 1220 LDA /AMPER.MONITOR
0307- 8D F7 03 1230 STA AMPERSAND.VECTOR+2
030A- 60 1240 RTS
1250 *-----
1260 AMPER.MONITOR
030B- C9 22 1270 CMP #$22 MUST BE QUOTATION MARK HERE
030D- D0 44 1280 BNE .6 SYNTAX ERROR
030F- 20 B1 00 1290 JSR AS.CHRGET
0312- A0 00 1300 LDY #0
0314- B1 B8 1310 .1 LDA (TXTPTR),Y
0316- F0 08 1320 BEQ .2
0318- 09 80 1330 ORA #$80
031A- 99 00 02 1340 STA MON.BUFFER,Y
031D- C8 1350 INY
031E- D0 F4 1360 BNE .1
0320- A9 8D 1370 .2 LDA #$8D
0322- 99 00 02 1380 STA MON.BUFFER,Y
0325- 98 1390 TYA
0326- 18 1400 CLC
0327- 65 B8 1410 ADC TXTPTR
0329- 85 B8 1420 STA TXTPTR
032B- 90 02 1430 BCC .25
032D- E6 B9 1440 INC TXTPTR+1
032F- 20 C7 FF 1450 .25 JSR MON.ZMODE
0332- 20 A7 FF 1460 .3 JSR MON.GETNUM
0335- 84 34 1470 STY MON.YSAV
0337- A0 17 1480 LDY #23
0339- 88 1490 .4 DEY
033A- 30 17 1500 BMI .6 SYNTAX ERROR
033C- D9 CC FF 1510 CMP MON.CHRTBL,Y
033F- D0 F8 1520 BNE .4 NOT THIS ENTRY
0341- C0 15 1530 CPY #21
0343- F0 08 1540 BEQ .5 <RETURN> ALONE
0345- 20 BE FF 1550 JSR MON.TOSUB
0348- A4 34 1560 LDY MON.YSAV
034A- 4C 32 03 1570 JMP .3
034D- 20 C5 FF 1580 .5 JSR MON.ZMODE-2
0350- 4C 00 FE 1590 JMP MON.BL1
0353- 4C C9 DE 1600 .6 JMP AS.SYNERR

```

Lines 1200-1240 link in the ampersand vector. This is the only part that would have to be changed if you move the routine.

When Applesoft sees an "&", it will JSR to AMPER.MONITOR. The A-register will hold the character following the "&", which we hope is a quotation mark. Lines 1270 and 1280 do this hoping.

Lines 1290-1380 copy the characters following the quotation mark into the monitor buffer starting at \$200. If you typed in the "&..." as a direct command, it is already in the monitor buffer but starts at \$202, so it gets shifted over two bytes. If the command is in a program, it will be copied out of program space into \$200. Applesoft has stripped off the sign bit from every byte, so my loop adds the sign bit back in to satisfy the monitor's requirements. Applesoft ends the line with a \$00 byte, and the monitor wants \$8D, so I fix that up too. I don't let colon terminate the line, because colon is a valid character in a monitor command line. I use "LDA (TXTPTR),Y" rather than repeated calls to AS.CHRGET because AS.CHRGET would eliminate blanks.

Lines 1390-1440 adjust the Applesoft pointer to the end of the line, so upon returning we won't get false syntax errors and the Applesoft program can continue executing.

Lines 1450-1590 parse the command line one command at a time, call on the monitor to execute each command, and finally return to Applesoft after the last command on the line. (The idea for this code came originally from code Steve Wozniak wrote for the mini-assembler in the old Apple monitor ROM.) Note that an illegal monitor command will result in a syntax error.

I thought it would now be possible to use the Amper-Monitor to write hex dumps on text files...BUT: Unfortunately DOS uses some critical zero page locations which prevent using the Amper-Monitor while writing on a text file. Monitor commands use locations \$3D through \$42, and so does DOS. I tried using the &"300.357 to do a hex dump into a text file, but DOS went wild and clobbered itself. Sorry, but I see no solution without changing DOS or recoding the entire monitor.

Yet Another New Version of DOS 3.3.....Bob Sander-Cederlof

In the July issue of AAL I outlined the changes Apple made to DOS 3.3 early this year. Today I received a new "Developer's System Master", with a cover letter claiming another correction to the APPEND routine. The letter binds developers to begin using the new version no later than November 1st.

If you like APPEND, or would like to like it, you might want to make these patches in your own system master. I am going to assume you already have the "early 1983" version, either because you bought a //e or a disk drive this year, or you copied one from a friend, or you made the patches from my July article. Here are the new changes:

"early 1983"	August, 1983
-----	-----
B683:4C 84 BA JMP \$BA84	B683:4C B3 B6 JMP \$B6B3
\$B6B3-B6CE:ALL ZEROES	B6B3:AD BD B5 LDA \$B5BD
	B6B6:8D E6 B5 STA \$B5E6
	B6B9:8D EA B5 STA \$B5EA
	B6BC:AD BE B5 LDA \$B5BE
	B6BF:8D E7 B5 STA \$B5E7
	B6C2:8D EB B5 STA \$B5EB
	B6C5:8D E4 B5 STA \$B5E4
	B6C8:BA TSX
	B6C9:8E 9B B3 STX \$B39B
	B6CC:4C 7F B3 JMP \$B37F
\$BA84-BA93:PATCH	BA84-BA93:ALL ZEROES

What Apple has done is move the patch they had put at \$BA84 down to \$B6B3 and added four extra lines to that patch. I HOPE IT IS NOW CORRECT!

D O W N L O A D I N G C U S T O M C H A R A C T E R S E T S

One of the features 'hidden' in many printers available today is their ability to accept user-defined character sets. With the proper software, these **custom characters** are 'downloaded' from your Apple II computer to the printer in a fraction of a second. Once the printer has 'learned' these new characters, they will be remembered until the printer is turned off.

After the downloading operation, you can use your printer with virtually any word processor. Just think of the possibilities! There's nothing like having your own **CUSTOM CHARACTERS** to help convey the message. And you still have access to those built-in fonts as well! **Here's a quick look at some possible variations:**

BUILT-IN

CUSTOM

10CPI: AaBbCcDdEeFfGgHhIiJjKk
12CPI: AaBbCcDdEeFfGgHhIiJjKk
17CPI: AaBbCcDdEeFfGgHhIiJjKk

AaBbCcDdEeFfGgHhIiJjKk
AaBbCcDdEeFfGgHhIiJjKk
AaBbCcDdEeFfGgHhIiJjKk

5CPI: AaBbCcDdEeFf
6CPI: AaBbCcDdEeFf
8CPI: AaBbCcDdEeFf

AaBbCcDdEeFf
AaBbCcDdEeFf
AaBbCcDdEeFf

And let's not forget Enhanced and Underlined printing as well...

AaBbCcDdEeFfGgHhIiJjKk
AaBbCcDdEeFfGgHhIiJjKk

AaBbCcDdEeFfGgHhIiJjKk
AaBbCcDdEeFfGgHhIiJjKk

The Font Downloader & Character Editor software package has been developed by RAK-WARE to help you unleash the power of your printer. The basic package includes the downloading software with 4 fonts to get you going. Also included is a character editor so that you can turn your creativity loose. Use it to generate unique character fonts, patterns, symbols and graphics. A detailed user's guide is provided on the program diskette.

SYSTEM REQUIREMENTS:

- * APPLE II, APPLE II Plus, APPLE //e or lookalike with 48K RAM
- * 'DUMB' Parallel Printer Interface Board (like Apple's Parallel Printer Interface, TYMAC's PPC-100 or equivalent)

The Font Downloader & Editor package is only \$39.95 and is currently available for either the Apple Dot Matrix Printer or C.Itoh 8510AP (specify printer). Epson FX-80 and OKiData versions coming soon. Enclose payment with order to avoid \$3.00 handling & postage charge.

R A K - W A R E
41 Ralph Road West Orange New Jersey 07052

Say You Saw It In **APPLE ASSEMBLY LINE!**

Base Address Calculation.....Bob Sander-Cederlof

I believe that Steve Wozniak was the first to use the tricks in a microcomputer, back in 1976 and 1977. All of the other designs I recall either used the more expensive static RAM, or used a complex circuit to refresh dynamic RAM arrays. Steve's design allowed the use of dynamic RAM without any separate circuitry for refresh.

Dynamic RAM needs refreshing because each bit cell is really only a capacitor, and the charge runs out after a few milliseconds. By reading each bit and re-writing it every few milliseconds, the data in memory is maintained as long as you like. Each 16384-bit RAM-chip is organized in 128 rows by 128 columns of bytes, and the chips are designed so that merely addressing each row often enough will keep the bits fresh as a daisy. Steve hooked up the Apple so that the process of keeping data displayed on the screen also ran through all the row addresses.

His second trick was to keep the screen (and therefore the RAM) happy without stealing any time from the CPU. He did this by using alternate half cycles of the clock. The one-megahertz clock runs the 6502 every other half cycle, and the screen gets its whacks at memory in between.

What has all the above to do with an article titled "Base Address Calculation"? Well, I'm getting to that. In order to address each row often enough, Steve re-arranged the address bits in a rather complicated way. As the screen is refreshed, scan-line by scan-line, bytes are read from RAM in an order that assures every RAM row is accessed about every 2 milliseconds. [For the exact details of this process, see Winston Gayler's "Apple II Circuit Description", pages 41-57.]

All this boils down to a need to go through a complicated calculation to convert a display line number into a base address in RAM. The process is implemented for the text screen at \$FBC1 in the monitor ROM; for the lo-res graphics screen at \$F847 in the monitor ROM; for the hi-res graphics screen at \$F417 in the Applesoft ROM.

If we represent the 8-bit value for the line number on the text screen as "000abcde", the base calculation computes the address in RAM for the first character on that line and stores the result in two bytes at \$28 and \$29 in the form "000001cd eabab000". The two bits "ab" may have values "00", "01", or "10" for lines 0-7, 8-15, and 16-23 respectively. The "abab000" part of the least significant byte of the base address represents "ab" times 40. Remember there are 40 characters on a line?

The hi-res base address calculation is more complicated, but it really the same thing. If we think of a text line as being made up of 8 hi-res lines, both calculations ARE the same. Except that the lo-res RAM starts at \$400, and the hi-res starts at \$2000. A hi-res line number runs from 0 through 191, or \$00 - \$BF. If we visualize it as "abcdefgh", the base

address calculation merely re-arranges the bits to "001fghcd eabab000". Note that if we multiply the text line number by 8 and run it through the hi-res calculation we will get "001000cd eabab000" which is correct except for starting at \$2000 rather than \$400.

The hi-res calculation inside Applesoft takes 33 bytes and 61 cycles. Harry Cheung, who lives in Onitsha, Nigeria, wrote a letter to Call APPLE (page 70, July, 1983) to present his shorter, faster version. Harry did it in 25 bytes and only 46 cycles (one more byte and 6 more cycles if you count the RTS, but I didn't count an RTS in the Applesoft version). Here is Harry's code, with my comments.

```

1200 *-----
1210 *      BASE ADDRESS CALCULATOR
1220 *      HARRY CHEUNG
1230 *      PMB 1601, ONITSHA, NIGERIA
1240 *      CALL APPLE, JULY 1983, PAGE 70
1250 *-----
081F- A8 1260 CALC TAY (TAY..TYA COULD BE PHA..PLA)
0820- 29 C7 1270 AND #$C7 ABCDEFGH
0822- 85 00 1280 STA 0 AB000FGH
0824- 09 08 1290 ORA #$08 FOR BASE = $2000, $10 FOR $4000
0826- 85 01 1300 STA 1 AB001FGH
0828- 98 1310 TYA ABCDEFGH
1320 *      CARRY..A-REG.....$00.....$01...
0829- 0A 1330 ASL A--BCDEFGH0 AB000FGH AB001FGH
082A- 0A 1340 ASL B--CDEFGH00 " "
082B- 66 00 1350 ROR 0 H-- " BAB000FG "
082D- 0A 1360 ASL C--DEFGH000 " "
082E- 26 01 1370 ROL 1 A-- " " B001FGHC
0830- 66 00 1380 ROR 0 G-- " ABAB000G "
0832- 0A 1390 ASL D--EFGH0000 " "
0833- 26 01 1400 ROL 1 B-- " " 001FGHCD
0835- 0A 1410 ASL E--FGH00000 " "
0836- 66 00 1420 ROR 0 G-- " EABAB000 001FGHCD
0838- 60 1430 RTS

```

I need to point out several things here. Harry used page zero locations \$00 and \$01 for the resulting base address. If you want to use his program with Applesoft, change them to \$26 and \$27. Harry save the line number temporarily in the Y-register. If the Y-register is already holding something important (it is in the Applesoft case), you can substitute PHA and PLA for the TAY and TYA above. Same number of bytes, but 3 cycles longer.

If you want REAL speed, and can spare a few more bytes, you need to pre-compute all the base addresses and store them in a table. Then you can use the line number as an index into the table and do a base address TRANSLATION in just a few cycles. For example, assume you store all the low-order bytes in a 192-byte table called LO.BASE, and similarly the high-order bytes at HI.BASE. If you get the line number in the Y-register, then you can convert the line number to a base address like this:

```

LDA LO.BASE,Y
STA $26
LDA HI.BASE,Y
STA $27

```

That takes 10 bytes of program, 384 bytes of table, and only 14 to 16 cycles. I say 14 to 16, because it depends on whether

either or both of the two tables cross page boundaries. If they each are entirely within a memory page, 14 cycles.

Now here is a little piece of code I wrote to test out Harry's calculator. It runs through each of the 192 lines and prints out the line number, an equal sign, the base address, and a space for each line (all in hex).

```

1010 *-----
1020 *      DRIVER ROUTINE TO PRINT OUT
1030 *      CALCULATED BASE ADDRESSES
1040 *-----
0800- A2 00 1050 TEST   LDX #0
0802- 8A      1060     TXA
0803- 20 1F 08 1070     JSR CALC
0806- 8A      1080     TXA
0807- 20 DA FD 1090     JSR $FDDA
080A- A5 01 1100     LDA 1
080C- 20 D3 FD 1110     JSR $FDD3
080F- A5 00 1120     LDA 0
0811- 20 DA FD 1130     JSR $FDDA
0814- A9 A0 1140     LDA #$A0
0816- 20 ED FD 1150     JSR $FDED
0819- E8      1160     INX
081A- E0 C0 1170     CPX #192
081C- 90 E4 1180     BCC .1
081E- 60      1190     RTS

```

The monitor address \$FDD3 is not a labelled entry point, but I think it will probably stay consistent in future editions of the Apple ROMs. It saves whatever is in the A-register, prints "=", restores the A-register, and falls into \$FDDA. The routine at \$FDDA prints the contents of A in hex.

Just for fun I also wrote some new versions of the text base address calculator. One of them is shorter but takes more time, and the other is longer but takes less time. Oh well, can't win every race! Here are listings of them both, followed by a commented listing of the Applesoft hi-res calculator.

```

1450 LRCALC.1
0839- 48      1460     PHA
083A- 29 18 1470     AND #$18      000DE000
083C- 0A      1480     ASL        000DE000
083D- 85 00 1490     STA 0
083F- 0A      1500     ASL        0DE00000
0840- 0A      1510     ASL        DE000000
0841- 05 00 1520     ORA 0        DEDE0000
0843- 85 00 1530     STA 0
0845- 68      1540     PLA        000DEFGH
0846- 4A      1550     LSR        0000DEFG
0847- 66 00 1560     ROR 0        HDEDE000
0849- 29 03 1570     AND #$03    000000FG
084B- 09 04 1580     ORA #$04    000001FG (FOR PAGE 1)
084D- 85 01 1590     STA 1
084F- 60      1600     RTS
1610 *-----
1620 LRCALC.2
0850- 48      1630     PHA
0851- 29 18 1640     AND #$18      000DE000
0853- F0 07 1650     BEQ .1
0855- C9 10 1660     CMP #$10
0857- A9 A0 1670     LDA #$A0
0859- B0 01 1680     BCS .1
085B- 4A      1690     LSR
085C- 85 00 1700 .1 STA 0        DEDE0000
085E- 68      1710     PLA        000DEFGH
085F- 4A      1720     LSR        0000DEFG
0860- 66 00 1730     ROR 0        HDEDE000
0862- 29 03 1740     AND #$03    000000FG
0864- 09 04 1750     ORA #$04    000001FG (FOR PAGE 1)
0866- 85 01 1760     STA 1
0868- 60      1770     RTS

```

```

1780 *-----
1790 *      FROM APPLESOFT ROM AT $F417-$F437
1800 *-----
0026- 1810 MON.GBASL .EQ $26
0027- 1820 MON.GBASH .EQ $27
00E6- 1830 HGR.PAGE .EQ $E6
      1840 AS.HRCALC
0869- 48 1850 PHA Y-POS ALSO ON STACK
086A- 29 C0 1860 AND #$C0 CALCULATE BASE ADDRESS FOR Y-POS
086C- 85 26 1870 STA MON.GBASL FOR Y=ABCDEFGH
086E- 4A 1880 LSR GBASL=ABAB0000
086F- 4A 1890 LSR
0870- 05 26 1900 ORA MON.GBASL
0872- 85 26 1910 STA MON.GBASL
0874- 68 1920 PLA
0875- 85 27 1930 STA MON.GBASH (C) (A) (GBASH) (GBASL)
0877- 0A 1940 ASL ?-ABCDEFGH ABCDEFGH ABAB0000
0878- 0A 1950 ASL A-BCDEFGH0 ABCDEFGH ABAB0000
0879- 0A 1960 ASL B-CDEFGH00 ABCDEFGH ABAB0000
087A- 26 27 1970 ROL MON.GBASH C-DEFGH000. ABCDEFGH ABAB0000
087C- 0A 1980 ASL A-DEFGH000 BCDEFGHC ABAB0000
087D- 26 27 1990 ROL MON.GBASH D-EFGH0000 BCDEFGHC ABAB0000
087F- 0A 2000 ASL B-EFGH0000 CDEFGHCD ABAB0000
0880- 66 26 2010 ROR MON.GBASL E-FGH00000 CDEFGHCD ABAB0000
0882- A5 27 2020 LDA MON.GBASH 0-FGH00000 CDEFGHCD EABAB000
0884- 29 1F 2030 AND #$1F 0-000FGHCD CDEFGHCD EABAB000
0886- 05 E6 2040 ORA HGR.PAGE 0-PPPFHCD CDEFGHCD EABAB000
0888- 85 27 2050 STA MON.GBASH 0-PPPFHCD PPFHGHCD EABAB000
      2060 *-----
088A- 60 2070 RTS
      2080 *-----

```

By the way, if you want to see the WHOLE thing...a commented listing of the entire Applesoft ROM, we have it on disk in format for the S-C Macro Assembler.

Saving Source with Apple's Mini-Assembler.....Jim Church
Trumbull, CT

I have discovered a way to store source code, complete with comments, on disk files for the Apple mini-assembler (at \$F666 in the Integer BASIC ROM or Language Card load). I use what I call "the world's best word processor", the one you get from S-C Software for \$50. I create a text file that looks like this:

```

FP
CALL-151
C080
F666G
300:LDX #C0 ;START WITH "A"-1
INX ;LOOP COMES HERE
TXA ;CHAR TO PRINT
JSR FDED ;PRINT IT
CPX #DA ;STOP AFTER "Z"
BCC 302 ;NOT THERE YET
RTS ;FINISHED!
FP
CALL768

```

Assuming I have Integer BASIC in my RAM card, EXECing the above text file assembles the code very nicely and even runs the program once! Note that the Mini-Assembler does allow comments following a ";".

Generic Screen Dump.....Steve Knouse
Tomball, TX

Some computer terminals have a special key on the keyboard which will dump whatever is on the screen to a printer. The following program will give the same function to an Apple, using the ctrl-P key.

Many different versions of screen dump programs have been written, and published hither and yon. Most of them work with the particular author's printer and interface combination, but not mine or yours. I found the one Bob S-C published in the July 81 issue of AAL to be like that, so I worked it over. Now I believe it can truly be called "generic", or at least general, because it runs on every combination of printers and interfaces I can find.

I tested it on systems using the following interfaces:

- Epson APL
- Orange Micro Grappler, Grappler+, & Buffered Grappler+
- Practical Peripherals Microbuffer II
- SSM AIO II & ASIO
- Tymac Parallel
- Videx PSIO

The screen dump should work with any interface which recognizes the Apple standard method for turning off video output. The standard is to "print" a control-I followed by an "N". Lines 2190 through 2250 perform the output of these two characters.

The only board I found which did not work with this convention was the SSM AIO board, so the program which follows has a special conditional assembly mode to make it assembly slightly different object code for that board. If you have that board, change line 1610 to say "VERSION .EQ AIO" and it will assembly your version. Instead of Lines 2190 through 2250 being assembled, lines 2260 through 2310 will. They do not show up in the listing, so here they are:

```
2260      .DO VERSION=AIO
2270      LDA #$80
2280      JSR COUT
2290      LDX SLOT
2300      STA NOVID,X
2310      .FIN
```

If your assembler does not support conditional assembly, you can merely type in the lines 2270-2300 above in place of lines 2190-2310.

If your printer interface is not plugged into slot 1, change the slot number in line 2030, or at \$0319.

Install the program by BRUNning the binary file of the object code, or by BLOADing it and doing a CALL768. Then whenever you type control-P, the screen will be printed. You can also call the screen dump from a running Applesoft program with CALL 794.

```

1000 *SAVE GENERIC SCREEN DUMP
1010 *-----
1020 *
1030 * GENERIC SCREEN DUMP
1040 *
1050 *-----
1060
0001- 1070 GENERIC .EQ 1
0002- 1080 AIO .EQ 2
1090
0001- 1100 VERSION .EQ GENERIC
1110
0024- 1120 CH .EQ $24
0028- 1130 BASL .EQ $28
0036- 1140 CSWL .EQ $36
0037- 1150 CSWH .EQ CSWL+1
0038- 1160 KSWL .EQ $38
0039- 1170 KSWH .EQ KSWL+1
1180
03EA- 1190 DOS.HOOK .EQ $3EA
1200
FBC1- 1210 BASCALC .EQ $FBC1
FD1B- 1220 COUT .EQ $FDED
FD1B- 1230 KEYIN .EQ $FD1B
FD0C- 1240 RDKEY .EQ $FD0C
FE95- 1250 OUTPORT .EQ $FE95
FC22- 1260 VTAB .EQ $FC22
1270
008D- 1280 CR .EQ $8D CARRIAGE RETURN
0578- 1290 NOVID .EQ $578
1300
1310 *-----
1320 .OR $300
1330 *-----
0300- A9 0B 1330 START LDA #ENTRY HOOK ROUTINE INTO DOS
0302- 85 38 1340 STA KSWL
0304- A9 03 1350 LDA /ENTRY
0306- 85 39 1360 STA KSWH
0308- 4C EA 03 1370 JMP DOS.HOOK
1380 *-----
030B- 20 1B FD 1390 ENTRY JSR KEYIN WAIT FOR A KEYPRESS
030E- C9 90 1400 CMP #$90 P ?
0310- D0 06 1410 BNE .1 NO
0312- 20 1A 03 1420 JSR DUMP YES
0315- 4C 0C FD 1430 JMP RDKEY
0318- 60 1440 .1 RTS
1450 *-----
0319- 01 1460 SLOT .DA #1
1470 *-----
031A- 48 1480 DUMP PHA SAVE A, X, Y
031B- 8A 1490 TXA
031C- 48 1500 PHA
031D- A8 1510 TAY
031E- 48 1520 PHA
031F- A5 24 1530 LDA CH SAVE CH
0321- 48 1540 PHA
0322- A5 36 1550 LDA CSWL SAVE OUTPUT HOOKS
0324- 48 1560 PHA
0325- A5 37 1570 LDA CSWH
0327- 48 1580 PHA
1590 *
0328- AD 19 03 1600 LDA SLOT COLD START BOARD
032B- 20 95 FE 1610 JSR OUTPORT IN SLOT 1
1620 .DO VERSION=GENERIC
032E- A9 89 1630 LDA #$89 KILL VIDEO ECHO
0330- 20 ED FD 1640 JSR COUT
0333- A9 CE 1650 LDA #"N"
0335- 20 ED FD 1660 JSR COUT
0338- EA 1670 NOP PAD TO STAY ALIGNED W/ AIO VERSION
1680 .FIN
1690 .DO VERSION=AIO
1740 .FIN
1750 *
0339- A9 8D 1760 LDA #CR START ON A' NEW LINE
033B- 20 ED FD 1770 JSR COUT
1780 *
033E- A2 00 1790 LDX #0 START W/ 1ST LINE (OTH)
0340- 86 24 1800 STX CH SET CH TO 0 SO PRINTER WON'T INDENT

```

0342- 8A	1810	TXA	LINE LOOP
0343- 20 C1 FB	1820 .1	JSR BASCALC	GET ADDR OF LINE
0346- A0 00	1830	LDY #0	START W/ 1ST CHARACTER (OTH)
0348- B1 28	1840	LDA (BASL),Y	GET A CHAR
034A- C9 A0	1850 .2	CMP #\$A0	CONVERT FLASH/INVERSE CHAR
034C- B0 04	1860 .3	BCS .4	NON-FLASHING U.C.
034E- 69 40	1870	ADC #\$40	..ALWAYS
0350- D0 F8	1880	BNE .3	MASK OFF HI BIT TO AVOID
0352- 29 7F	1890	AND #\$7F	EPSON BLOCK GRAPHICS
	1900 .4		PRINT IT
	1910		LOOP FOR ANOTHER CHAR
0354- 20 ED FD	1920	JSR COUT	
0357- C8	1930	INY	
0358- C0 28	1940	CPY #40	
035A- 90 EC	1950	BCC .2	
035C- A9 8D	1960	LDA #CR	END OF LINE
035E- 20 ED FD	1970	JSR COUT	LOOP FOR ANOTHER LINE
0361- E8	1980	INX	
0362- E0 18	1990	CPX #24	
0364- 90 DC	2000	BCC .1	
	2010		
0366- 68	2020	PLA	RESTORE OUTPUT HOOKS
0367- 85 37	2030	STA CSWH	
0369- 68	2040	PLA	
036A- 85 36	2050	STA CSWL	
036C- 68	2060	PLA	RESTORE CH
036D- 85 24	2070	STA CH	
036F- 20 22 FC	2080	JSR VTAB	AND LINE
0372- 68	2090	PLA	RESTORE Y, X, A
0373- A8	2100	TAY	
0374- 68	2110	PLA	
0375- AA	2120	TAX	
0376- 68	2130	PLA	
0377- 60	2140	RTS	..THAT'S ALL FOLKS

ES-CAPE will set your creativity free!

ES-CAPE will help you develop, enter, and modify Applesoft programs. Even if you are only copying a program from a magazine, ES-CAPE will help you do it three times faster!

Visualize this: by pressing just a key or two, you can...

- See the disk catalog, select a program, and load it into memory.
- Browse through the program a screen or a line at a time.
- Edit lines using powerful commands like the word processors have: insert, delete, truncate, overtype, scan to beginning or end or to a particular character, and more.
- See the values of the variables used by your Applesoft program as it ran.
- Save the modified program. ES-CAPE remembers the file name for you!

ES-CAPE is easy to learn and use!

- Well-written User Manual guides you through the learning process.
- Handy Quick Reference Card reminds you of all features and commands.
- Built-in help screens and menus refresh your memory. You don't have to memorize anything!
- The disk is NOT protected! You can put ES-CAPE on every disk you own, and make as many backup copies as you need.

ES-CAPE will speed up and simplify your Applesoft programming!

- Choose a starting value and step size for automatic line numbering.
- Swiftly find all references to a given variable, line number, or any other sequence of characters.
- Quickly and automatically scan your program for any sequence of characters and replace them with a new spelling.
- Enter commonly used words or phrases with a single keystroke. A full set of pre-defined macros is provided, which you may modify as you wish.
- Display a DOS Command Menu with a single keystroke. A second keystroke selects CATALOG, LOAD, SAVE, and other common DOS commands. You can easily manage a disk-full of programs!

ES-CAPE is available now at many fine computer stores, or directly from S-C Software Corporation. The price is only \$60.

S-C SOFTWARE CORPORATION
2331 Gus Thomasson, Suite 125
Dallas, TX 75228 (214) 324-2050

Professional Apple Software Since 1978

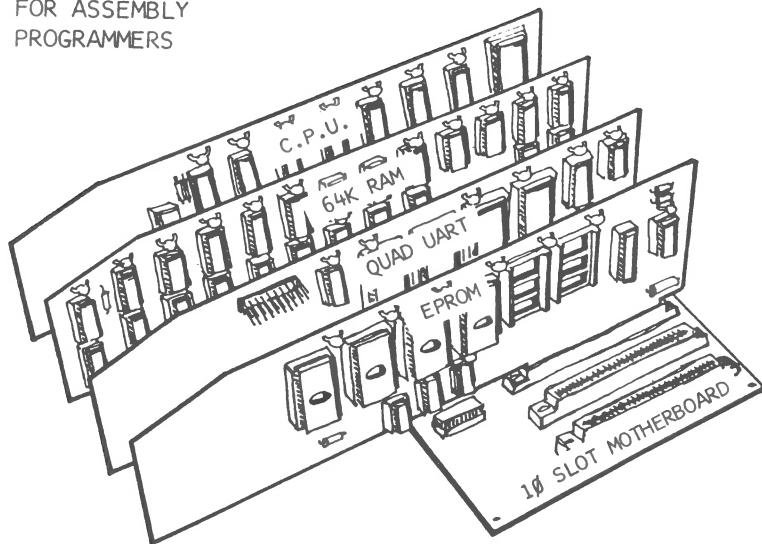
Visa, MasterCard, American Express, COD accepted.

Apple is a trademark of Apple Computer Inc.



APPLESEED

HARDWARE FOR ASSEMBLY
LANGUAGE PROGRAMMERS



Appleseed boards allow systems to be designed and programmed on a standard Apple Computer and then produced with a board set which costs a fraction of what an Apple Computer would cost. This approach allows the use of a multitude of sophisticated programming and hardware tools which have been designed for use on the Apple Computer. Low cost, small size, low power consumption, and flexibility are the advantages with **Appleseed**. All of the Apple bus conventions are preserved which allows the use of almost any custom board which has been designed for use in the Apple computer. Programs are developed using a good editor/assembler (such as S-C Macro); then after the system is debugged and running properly, a set of EPROMs are blown and inserted into the destination machine. Soon we will have available a set of boards which will allow direct loading of the program from your Apple computer into the **Appleseed** for the testing of the program prior to actual blowing of a set of EPROMs.

The **Appleseed** approach allows the greatest possible flexibility in system design. You can pick and choose among the various **Appleseed** components as well as a multitude of peripheral boards made by others in order to design your own customized system. The only limitation is that we don't supply firmware and we don't support the standard Apple CRT graphics. Flexibility is the keyword! You buy only what you need for your application.

Optional packaging is also available. We have a "cute little box with walnut side panels" or a 19 inch rack mount with built-in disk drives.

Appleseed products are not available through computer stores. For more information call or write us directly. Also look for us at the Applefest in San Francisco, October 28th-30th.

Apple is a registered trademark of Apple Computer Inc.

DOUGLAS ELECTRONICS - 718 MARINA - SAN LEANDRO, CA 94577 - (415)483-8770

New CATALOG Interrupt.....Col. Paul L. Shetler
Tripler AMC, Hawaii

Most of the routines I've seen to terminate a CATALOG listing involve patching in a routine that checks for a particular key input and adding code to do different actions, like aborting or single-stepping the catalog list. Here is a modification I came up with that requires only a small change and no additional code.

This is the section of DOS that handles a new line in the CATALOG display:

```

                1000          .OR $AE2C
                1010
AE2C- 4C 7F B3 1020      JMP $B37F      leave File Manager
AE2F- A9 8D 1030 NEWLN   LDA #$8D      carriage return
AE31- 20 ED FD 1040      JSR $FDED      MON.COUT
AE34- CE 9D B3 1050      DEC $B39D      line count
AE37- D0 08 1060          BNE .1
AE39- 20 0C FD 1070      JSR $FD0C      MON.RDKEY
AE3C- A9 15 1080          LDA #$15      count 21 lines
AE3E- 8D 9D B3 1090      STA $B39D      reset line count
AE41- 60 1100 .1         RTS
```

Line 1020 is really the end of the previous routine, but we're going to be borrowing it, so I'll show it here. NEWLN is called every time the catalog list finishes a file name.

Notice that two bytes are wasted in lines 1030-1040. Why do LDA #\$8D, JSR \$FDED, when JSR \$FD8E does the same thing? Two bytes may not sound like much, but in this case it's enough to work some magic! Try replacing the above piece of DOS with this:

```

                1000          .OR $AE2C
                1010
AE2C- 4C 7F B3 1020 EXIT  JMP $B37F      leave File Manager
AE2F- 20 8E FD 1030 NEWLN JSR $FD8E      MON.CROUT
AE32- CE 9D B3 1040      DEC $B39D      line count
AE35- D0 0A 1050          BNE .1         return if not done
AE37- 20 0C FD 1060      JSR $FD0C      get a keypress
AE3A- 29 17 1070          AND #$17      the magic number
AE3C- F0 EE 1080          BEQ EXIT       abort CATALOG
AE3E- 8D 9D B3 1090      STA $B39D      new line count
AE41- 60 1100 .1         RTS
```

Slipping in that AND #\$17, BEQ EXIT, has several effects:

1. Space Bar or Back Arrow will terminate the listing.
2. Forward Arrow will advance the listing one page (just like normal.)
3. The "A" key will advance the listing one line.

And it all fits into the original space! The other keys will have different effects, depending on the value left in the accumulator after AND #\$17. Most keys will advance the listing between 1-23 lines.

Try substituting other values for the \$17 in line 1070. Remember that the value of (Keypress AND Value) will be the new line count. The catalog display will scroll up by that number of lines. If the result is zero, the catalog display will end. The maximum result is the same as the mask value, that is, 23 lines for a \$17 mask.

[My favorite mask value is \$4F. With that value SPACE still breaks the display, but now the numeral keys scroll up by the same number of lines, i.e., pressing the "1" key gives one more line, "2" shows two more names, and so on. Also, the "0" (oh, not zero) key scrolls up by 79 lines, which usually means all the way to the end of the catalog....Bill]

80 Column ASCII Monitor Dump.....Mike Dobe
O'fallon, IL

I have been trying out the monitor patches in the July issue of AAL for adding an ASCII display to the memory dump, and I have two problems with them. Because the routines place the characters directly into the Apple's screen memory, they do not work with my 80 column card. The same problem also arises when I want to send a dump to a printer. As a solution to this problem I present still another monitor patch for an ASCII display. My version is slightly longer than the others, but it still fits in the cassette tape portion of the monitor (just barely, I might add).

In order to take advantage of the 80 column display I first made the following patches to the monitor:

FDA6:0F
FDB0:0F

These changes allow the dump routine to print 16 values on each line, rather than the usual eight.

Since the characters have to be printed after the current line of the dump is finished, I need a place to buffer up to 16 characters. \$BCDF, an unused area in DOS, serves this purpose. My routine buffers each byte before calling PRBYTE to display the hex value. If a particular byte will be the last one on that line of the dump, the patch calls PRBYTE to print the byte, then tabs to column 60 and displays the contents of the buffer. Upper and lower case characters are printed as they are, and control characters are replaced with blanks. (That's my style. As Bob said in July, choose your own favorite!)

Of course the following patch needs to be made to the dump code, to call my routine (this is the same as shown in the July article):

FDBE:C9 FC

The patch can be used with a 40 column display by ignoring the above patches to \$FDA6 and \$FDB0, and by making the following changes to my patch routine:

```

1140      AND #7
1200      EOR #7
1300      LDA #30
1420      CPX #8

```

This patch was tested on a Microtek Magnum 80 card, but it should work on other brands as well.

[It also works fine with the STB80 card, and the Apple
//e...Bill]

```

1000 *SAVE S.MON ASCII DISPLAY (DOBE)
1010 *-----
0024- 1020 CH      .EQ $24
003C- 1030 A1L    .EQ $3C
003D- 1040 A1H    .EQ $3D
003E- 1050 A2L    .EQ $3E
003F- 1060 A2H    .EQ $3F
BCDF- 1070 BUFFER .EQ $BCDF
FDDA- 1080 PRBYTE .EQ $FDDA
FDED- 1090 COUT   .EQ $FDED
1100 *-----
1110      .OR $FCC9
1120      .TA $CC9
1130
FCC9- 48      1140 PATCH PHA      save byte
FCCA- A5 3C   1150      LDA A1L   low byte of dump address
FCCC- 29 0F   1160      AND #$F    is transformed to
FCEE- AA      1170      TAX      offset in buffer
FCCF- 68      1180      PLA      get original byte back
FCD0- 48      1190      PHA      but keep it on the stack
FCD1- 9D DF BC 1200      STA BUFFER,X buffer the character
FCD4- E0 0F   1210      CPX #$F    last byte of line?
FCD6- F0 0C   1220      BEQ .0     if so, print the buffer
FCD8- A5 3E   1230      LDA A2L
FCDA- C5 3C   1240      CMP A1L    done with range?
FCDC- D0 29   1250      BNE .3     return to monitor if not
FCDE- A5 3F   1260      LDA A2H
FCE0- C5 3D   1270      CMP A1H    check high bytes
FCE2- D0 23   1280      BNE .3     return if more
1290
FCE4- 68      1300 .0      PLA
FCE5- 20 DA FD 1310      JSR PRBYTE print the last byte
FCE8- A9 3C   1320      LDA #60    tab to column 60
FCEA- 85 24   1330      STA CH
FCEC- A2 00   1340      LDX #0
FCEE- BD DF BC 1350 .1      LDA BUFFER,X display the buffer
FCF1- 09 80   1360      ORA #$80
FCF3- C9 A0   1370      CMP #$A0    control character?
FCF5- B0 02   1380      BCS .2
FCF7- A9 A0   1390      LDA #$A0    if so, substitute blank
FCF9- 20 ED FD 1400 .2      JSR COUT   print the character
FCFC- A9 A0   1410      LDA #$A0
FCFE- 9D DF BC 1420      STA BUFFER,X blank out buffer as we go
FD01- E8      1430      INX
FD02- E0 10   1440      CPX #$10    done?
FD04- 90 E8   1450      BCC .1     no, go on
FD06- 60      1460      RTS
1470
FD07- 68      1480 .3      PLA      restore original byte
FD08- 4C DA FD 1490      JMP PRBYTE returns to caller

```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$15 per year in the USA, sent Bulk Mail; add \$3 for First Class postage in USA, Canada, and Mexico; add \$13 postage for other countries. Back issues are available for \$1.50 each (other countries add \$1 per back issue for postage).

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)